HAL
open science

Be Scalable and Rescue My Slices During Reconfiguration

Adrien Gausseran, Frédéric Giroire, Brigitte Jaumard, Joanna Moulierac

To cite this version:

Adrien Gausseran, Frédéric Giroire, Brigitte Jaumard, Joanna Moulierac. Be Scalable and Rescue My Slices During Reconfiguration. [Research Report] Inria - Sophia Antipolis; I3S, Université Côte d'Azur; Concordia University. 2019. hal-02416096

HAL Id: hal-02416096

https://inria.hal.science/hal-02416096v1

Submitted on 17 Dec 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Be Scalable and Rescue My Slices During Reconfiguration

Adrien Gausseran, Frédéric Giroire, Brigitte Jaumard, Joanna Moulierac

RESEARCH REPORT
N° XXXX
December 2019
Project-Team Coati

ISSN 0249-6399 ISRN INRIA/RR--XXXX--FR+ENG

Be Scalable and Rescue My Slices During Reconfiguration

Adrien Gausseran*, Frédéric Giroire*, Brigitte Jaumard†, Joanna Moulierac*

Project-Team Coati

Research Report n° XXXX — December 2019 — 27 pages

Abstract: Modern 5G networks promise more bandwidth, less delay, and more flexibility for an ever increasing number of users and applications, with Software Defined Networking, Network Function Virtualization, and Network Slicing as key enablers. Within that context, efficiently provisioning network and cloud resources of a wide variety of applications with dynamic users' demands is a real challenge. In this work, we consider the problem of network slice reconfiguration. Reconfiguring regularly network slices allows to reduce the network operational cost. However, it impacts users' *Quality of Service* by changing the routing. To solve this issue, we study solutions implementing a *make-before-break* scheme. We propose new models and scalable algorithms (relying on column generation techniques) that solve large data instances in few seconds.

Key-words: Reconfiguration, Software Defined Networking, Service Function Chains, Network Function Virtualization

This work has been supported by the French government, through the UCA JEDI and EUR DS4H Investments in the Future projects managed by the National Research Agency (ANR) with the reference number ANR-15-IDEX-01 and ANR-17-EURE-004 and by Inria associated team EfDyNet.

* Université Côte d'Azur, CNRS, I3S, Université de Nice, Inria Sophia Antipolis, France
† University of Concordia, Montreal, Canada

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Sauvez mes slices pendant la reconfiguration

Résumé : Les réseaux 5G modernes promettent plus de bande passante, moins de délai et plus de flexibilité pour un nombre toujours croissant d'utilisateurs et d'applications, avec la programmation logicielle des réseaux (SDN), la virtualisation des fonctions réseau, le découpage du réseau en slices comme facteurs clés de réussite. Dans ce contexte, le provisionnement efficace des ressources réseau d'une grande variété d'applications avec les demandes des utilisateurs dynamiques est un véritable défi. Dans ce travail, nous considérons le problème de la reconfiguration des slices réseau. La reconfiguration régulière des slices permet de réduire le coût opérationnel du réseau. Cependant, il a un impact sur le *Qualité de service* des utilisateurs en modifiant le routage. Pour résoudre ce problème, nous étudions des solutions mettant en oeuvre un schéma de reconfiguration des slices avec une technique *make-before-break*. Nous proposons de nouveaux modèles et algorithmes s'appuyant sur des techniques de génération de colonnes qui permettent de résoudre de grandes instances du probème en quelques secondes.

Mots-clés : Reconfiguration, SDN (réseaux logiciels), SFC (chaînes de service), NFV (fonctions réseaux virtuelles)

1 Introduction

The Network Function Virtualization (NFV) paradigm is a major technology of 5G networks. Over the past decade, it has been widely deployed and a large number of studies investigated its use and benefits. Its core principle is to break the dependence on dedicated hardware like traditional expensive middleboxes by allowing network functions (e.g., firewall, load balancing, Virtual Private Network (VPN) gateways, content filtering) to be virtualized and implemented in software, and executed on generic servers. Virtual Network Functions (VNFs) can be instantiated and scaled on demand without the need to install new equipment, increasing flexibility with user demands [1]. In parallel, we also saw the emergence of Software-Defined Networking (SDN) that simplifies network monitoring and management. By decoupling the control plane from the data plane and abstracting network intelligence into a central controller, SDN allows a global vision and control of the network [2]. Combination of SDN and NFV leads to dynamic, programmable and flexible networks in which the network infrastructure and resources are shared between network services.

The 5G technology is envisioned to allow a multi-service network supporting a wide range of communication scenarios with a diverse set of performance and service requirements. The concept of network slicing has been proposed to address these diversified service requirements. A network slice is an end-to-end logical network provisioned with a set of isolated virtual resources on a shared physical infrastructure [3, 4]. Moreover, slicing allows an efficient usage of resources, as VNFs can be instantiated and released on demand by slices. Besides, slices can be deployed whenever there is a service request, reducing the network operator costs [4]. With all these key features, Network slicing will thus be a fundamental feature of 5G networks [3].

Dynamic resource allocation is one of the key challenge of network slicing. In a dynamic scenario, the network state changes continuously due to the arrival and departure of flows. As the granting of new flows is done without impacting the ongoing ones, we may end up with a fragmented provisioning, and thus with an inefficient resource usage. Therefore, network operators must adjust network configurations in response to changing network conditions to fully exploit the benefits of the SDN and NFV paradigms, and to minimize the operational cost (e.g., software licenses, energy consumption, and Service Level Agreement (SLA) violations).

We here consider the problem of both rerouting traffic flows and improving the mapping of network functions onto nodes in the presence of dynamic traffic, with the objective of bringing the network back to a close to optimal operating state, in terms of resource usage. Rerouting demands and migrating VNFs take several steps. Usually, network carriers/operators cannot afford traffic interruption, due to their SLAs, as it may have a non-negligible impact on the Quality of Service (QoS) experienced by the users. Their strategy is then to perform the reconfiguration by using a two–phase approach. First, a new route is established while keeping the initial one enabled (i.e., two redundant data streams are both active in parallel). Then, the transmission moves on the new route and the resources used by the initial one are released. This strategy is often referred to as *make-before-break*. In this work, to the best of our knowledge, we are the first to propose scalable models to reconfigure network slices while implementing such mechanisms to avoid QoS degradation.

Our contributions in this paper are as follows:

- We propose an Integer Linear Program (`slow-rescue`) to reconfigure, with a *make-before-break mechanism*, the routing and provisioning of a set of slices.

- We propose two *scalable* models, `rescue-ILP` and `rescue-LP`, with `rescue` standing for "REconfiguration of network Slices with ColUmn gEneration without interruption". Both are based on a decomposition model and are solved using column generation. Our algo-

rithms reconfigure a given set of network slices from an initial routing and placement of network functions to another solution that reduces the network operational cost. Our solutions scale on large networks as we succeeded in solving data instances with 65 nodes and 108 links, and a hundred of network slices in few seconds, a lot faster than with a classic compact Integer Linear Program (ILP) formulation.

- We show that our solutions allow *the decrease of the network cost* without degrading the QoS (as the network slices are not interrupted thanks to the make-before-break approach) in moderate running times. Moreover, we can accept more network slices when the network is congested compared to solutions without any reconfiguration.

2 Related Work

In the last years, a large corpus of works has studied the deployment and management of network services, see [5] and [6] for surveys. In particular, the problem of jointly routing demands and provisioning their needed VNFs has attracted a lot of attention. A large number of efficient algorithms and optimization models have been proposed in order to minimize setup cost [7, 8] or take into account the chaining constraints [9, 10]. Most of these works have only considered scenarios in which, when a service is deployed, its route and used virtual resources are not changed during its lifetime. However, the churn of network services makes that even an optimal service deployment may lead to a sub-optimal use of network resources after some time, when some services have left.

Inspired by the classic defragmentation mechanism in optical networks [11], it has been proposed to carry out reconfigurations of network and virtual resources regularly in order to bring the network closer to an optimal state of operation. The goals can be diverse: optimizing network usage, granting more requests, modifying the capacities of flows already allocated on the network or even to overcome network failures.

The readjustment of Service Function Chains (SFCs) has been studied in [12]. The authors formulate an ILP and a column generation model in order to jointly optimize the deployment of the SFCs of new users and the readjustment of the SFCs already provisioned in the network while considering the trade-off between resource consumption and operational overhead. [13] studies the trade-off between the reconfiguration of SFCs and the optimality of the reconfigured routing and placement solution.

Gao and Rouskas [14] considered the reconfiguration of virtual networks. They proposed online algorithms to minimize the maximum utilization of substrate nodes and links while bounding the number of virtual nodes that have to be migrated.

Recently, the problem has been studied for network slices. [15] proposes a hybrid slice reconfiguration mechanism. The goal of the authors is to optimize the profit of a network slice provider, i.e., the total utility gained by serving slices minus the resource consumption and reconfiguration cost. The reconfiguration overhead of a slice includes two aspects: service interruption and reconfiguration resource cost.

Similarly, all works on reconfigurations of virtual resources (virtual networks, slices or service function chains) include a cost expressing the degradation of the client's QoS. On the contrary, our goal is to *avoid this QoS degradation* by proposing a *make-before-break* mechanism, in which the new route is reserved and the new virtual resources are installed before the slice is reconfigured. A similar mechanism has been proposed in [16]. However, we are the first to propose a scalable decomposition model based on column generation to solve it.

$G = (V, E)$	Network: V represents the node set and E the link set.
C_ℓ	Bandwidth link capacity of $\ell \in E$.
Γ_ℓ	Link delay of $\ell \in E$.
C_u	Resource node capacity (e.g., CPU, memory, and disk) of node $u \in V$.
Δ_f	Number of bandwidth units required by function $f \in F$.
$c_{u,f}$	Usage cost of function $f \in F$, which also depends on node u.

Each demand $d \in D$ is modeled by a quintuplet :

(v_s, v_d)	Source and destination nodes,
c_d	Ordered network function sequence,
BW_d	Required bandwidth units,
γ_d	Maximum required delay for the slice.

Table 1: Notations

3 Problem Statement and Notations

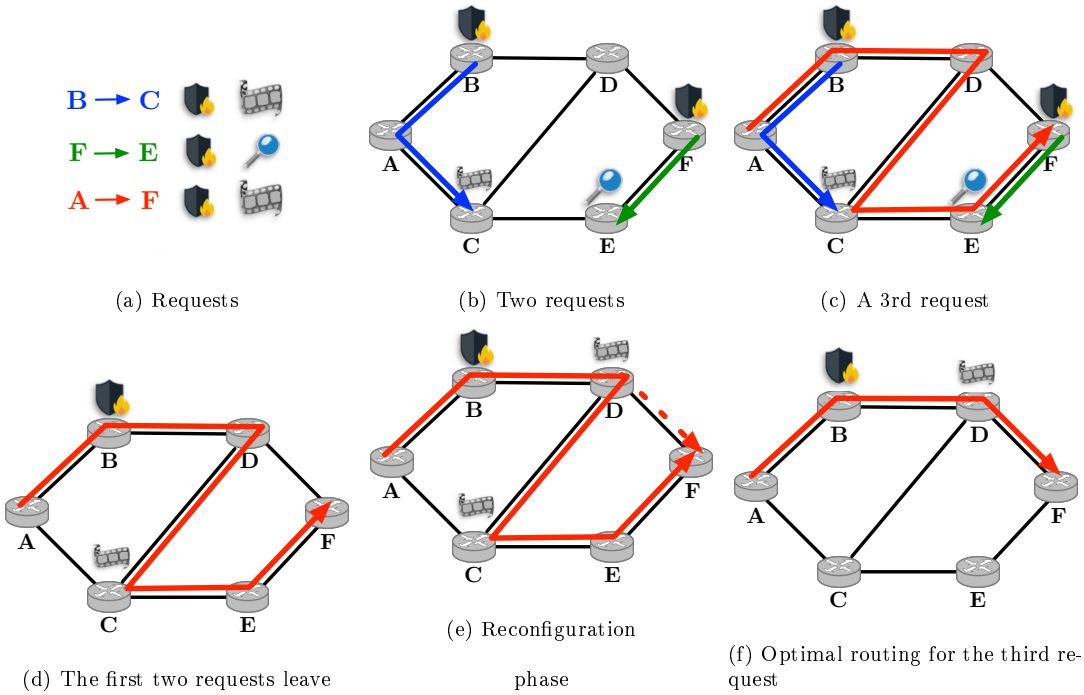
We consider the network as a directed capacitated graph $G = (V, E)$ where V represents the node set and E the link set. C_u is the resource node capacity (e.g., CPU, memory, and disk) of node $u \in V$. C_ℓ is the bandwidth link capacity and Γ_ℓ is the link delay of link $\ell \in E$. $t \in \{1, T\}$ is the number of steps used for the reconfiguration. Δ_f is the number of bandwidth units required by function $f \in F$.

A slice can be modeled by a set of demands following for example [17, 18]. Each demand $d \in D$ is modeled with a quintuplet: v_s the source, v_d the destination, c_d the ordered sequence of network functions that need to be performed, BW_d the required units of bandwidth, and γ_d the delay requirement.

Table 1 summarizes the notations used throughout the paper.

In a dynamic scenario with no information on future traffic, each demand is routed individually while *minimizing the network operational cost* defined by the weighted sum of link bandwidth and VNF usage costs (licenses, energy consumption, etc). As requests come and leave over time, allocations that are locally optimal at a given instant can bring the network in a global sub-optimal state. Our goal is to reconfigure the network to improve resource usage and therefore the operational costs. In doing this, we use the *make-before-break* mechanism to avoid network service disruption due to traffic rerouting.

Example. Figure 1 illustrates an example for the reconfiguration of a request using a *make-before-break* process. When the request from A to F arrives, two requests have already been routed (step (b)). To avoid the usage cost of new VNFs, the route from A to F with minimum cost is a long 5–hops route (step (c)). When requests from B to C and from F to C leave, the request is routed on a non-optimal path (step (d)) which uses more resources than necessary. We compute one optimal 3-hop path and reroute the request on it (step (f)) with an intermediate make-before-break step (step (e)) in which both routes co-exist. In this example, the reconfiguration can be done with only one step of reconfiguration, but we will consider in the following up to 3 steps of reconfiguration.



(a) Requests (b) Two requests (c) A 3rd request

(d) The first two requests leave (e) Reconfiguration phase (f) Optimal routing for the third request

Figure 1: An example of the reconfiguration of a request using a *make-before-break* approach with one step.

3.1 Layered graph

Similarly as in [9], our models are based on the concept of a layered graph. In order to model the chaining constraint of a demand, we associate to each demand d a layered graph $G^L(d)$. We denote by $u_{i,l}$ the copy of node u_i in layer l. The path for demand d starts from node $v_{s,0}$ in layer 0 and ends at node $v_{d,|c_d|}$ in layer $|c_d|$ where $|c_d|$ denotes the number of VNFs in the chain of the demand.
Given a link (u_i, v_j), each layer l has a link $(u_{i,l}, v_{j,l})$ defined. This property does not hold for links of the kind $(u_{i,l}, u_{i,l+1})$. Indeed, a node may be enabled to run only a subset of the virtual functions. To model this constraint, given a demand d we add a link $(u_{i,l}, u_{i,l+1})$ only if node u is enabled to run the $(l+1)-th$ function of the chain of d. The $l-th$ function of the chain of d will be denoted by $f_l^{c_d}$.
A path on the layered graph corresponds to an assignment to a demand of both a path and the locations where functions are being run. Using a link $(u_{i,l}, v_{j,l})$ on G^L, implies using link ℓ on G. On the other hand, using link $(u_{i,l}, u_{i,l+1})$ implies using the $(l+1)-th$ function of the chain at node u. Capacities of both nodes and links are shared among layers.

See Figure 2 for an example of a graph with three layers.

4 Optimization models

This section described first the compact ILP model (`slow-rescue`) to solve our problem, and then, the two models (`rescue-ILP` and `rescue-LP`) based on column generation methods.

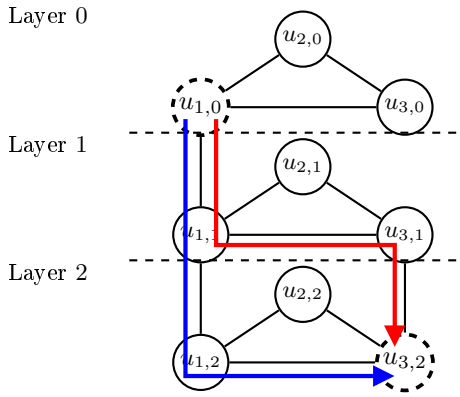


Figure 2: The layered network $G^L(d)$ associated with a demand d such that $v_s = u_1$, $v_d = u_3$, and $c_d = f_1, f_2$, within a triangle network. f_1 is allowed be installed on u_1 and f_2 on u_1 and u_3. Source and destination nodes of $G^L(d)$ are $u_{1,0}$ and $u_{3,2}$. Two possible SFCs that satisfy d are drawn in red (f_1 is in u_1, f_2 in u_3) and blue (f_1 and f_2 are in u_1).

4.1 ILP Model: `slow-rescue`

The compact ILP model, `slow-rescue`, is an Integer Linear Program based on the notion of layered graph described previously.

Variables:
• $\varphi^{d,t}_{\ell,i} \in [0,1]$ is the amount of flow on Link ℓ in Layer i at time step t for Demand d.
• $\alpha^{d,t}_{u,i} \in [0,1]$ is the amount of flow on Node u in Layer i at time step t for Demand d.
• $x^{d,t}_{\ell,i} \in [0,1]$ is the maximum amount of flow on Link ℓ in Layer i at time steps t and $t-1$ for Demand d.
• $y^{d,t}_{u,i} \in [0,1]$ is the maximum amount of flow on Node u in Layer i at time steps t and $t-1$ for Demand d.
• $\omega^{d,t} \in [0,1]$, where $\omega^{d,t} = 0$ if the allocation of demand d is modified between time steps t or $t-1$.
• $z_{u,f} \in [0,1]$, where $z^f_u = 1$ if function f is activated on Node u at time step T in the final routing.

The optimization model starts with the initial configuration as an input. Thus, for each demand $d \in D$, at initial time step 0, the variables $\varphi^{d,0}_{\ell,i}$ (for each link $\ell \in E$, layer $i \in \{0, ..., |c_d|\}$) and $\alpha^{d,0}_{u,i}$ (for each node $u \in V$, layer $i \in \{0, ..., |c_d|\}$) are known.

Objective: minimize the amount of network resources consumed during the last reconfiguration time step T.

$$\min \sum_{d \in D} \sum_{\ell \in E} \sum_{i=0}^{|c_d|} \mathrm{BW}_d \, \varphi^{d,T}_{\ell,i} + \beta \sum_{u \in V} \sum_{f \in F} c_{u,f} \, z_{u,f} \tag{1}$$

Constraints:
Flow conservation constraints. For each demand $d \in D$, node $v \in V$, time step $t \in \{1, ..., T\}$.

$$\sum_{\ell \in \omega^+(u)} \varphi^{d,t}_{\ell,0} - \sum_{\ell \in \omega^-(u)} \varphi^{d,t}_{\ell,0} + \alpha^{d,t}_{u,0} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \tag{2}$$

$$\sum_{\ell \in \omega^+(v)} \varphi^{d,t}_{\ell,|c_d|} - \sum_{\ell \in \omega^-(v)} \varphi^{d,t}_{\ell,|c_d|} - \alpha^{d,t}_{u,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{3}$$

$$\sum_{\ell \in \omega^+(u)} \varphi^{d,t}_{\ell,i} - \sum_{\ell \in \omega^-(u)} \varphi^{d,t}_{\ell,i} + \alpha^{d,t}_{u,i} - \alpha^{d,t}_{u,i-1} = 0$$

$$0 < i < |c_d| \tag{4}$$

Node usage over two consecutive time periods. For $d \in D$, $u \in V$, $i \in \{0, ..., |c_d| - 1\}$, $t \in T$.

$$\alpha^{d,t}_{u,i} \leq y^{d,t}_{u,i} \tag{5}$$

$$\alpha^{d,t-1}_{u,i} \leq y^{d,t}_{u,i} \tag{6}$$

$$\alpha^{d,t}_{u,i} + \alpha^{d,t-1}_{u,i} - \omega^{d,t} \leq y^{d,t}_{u,i}. \tag{7}$$

Link usage over two consecutives time periods. For $d \in D, \ell \in E$, Layer $i \in \{0, ..., |c_d|\}$, $t \in \{1, ..., T\}$.

$$\varphi^{d,t}_{\ell,i} \leq x^{d,t}_{\ell,i}$$

$$\varphi^{d,t-1}_{\ell,i} \leq x^{d,t}_{\ell,i}$$

$$\varphi^{d,t}_{\ell,i} + \varphi^{d,t-1}_{\ell,i} - \omega^{d,t} \leq x^{d,t}_{\ell,i}$$

Make Before Break - Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed C_u considering the resources used over two consecutives time periods. For each Node $u \in V$, time step $t \in \{1, ..., T\}$.

$$\sum_{d \in D} \mathrm{BW}_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \, y^{d,t}_{u,i} \leq C_u \tag{8}$$

Make Before Break - Link capacity constraints. The capacity of a link $\ell \in E$ is shared between each layer and cannot exceed C_ℓ considering the resources used over two consecutive time periods. For $\ell \in E$, $t \in \{1, ..., T\}$.

$$\sum_{d \in D} \mathrm{BW}_d \sum_{i=0}^{|c_d|} x^{d,t}_{\ell,i} \leq C_\ell \tag{9}$$

Delay constraint. The sum of the delays of all links traversed by the flow of a demand d must not exceed the maximum delay accepted by the demand. For $d \in D$, $t \in \{1, ..., T\}$

$$\sum_{i=0}^{|c_d|} x^{d,t}_{\ell,i} \, \Gamma_\ell \leq \gamma_d \tag{10}$$

Functions activation. To know which functions are activated on which nodes in the final routing. For $u \in V$, $f \in F$, $d \in D$, and $i \in \{0, ..., |c_d| - 1\}$,

$$\alpha^{d,T}_{u,i} \leq z_{u,f_i^{c_d}} \tag{11}$$

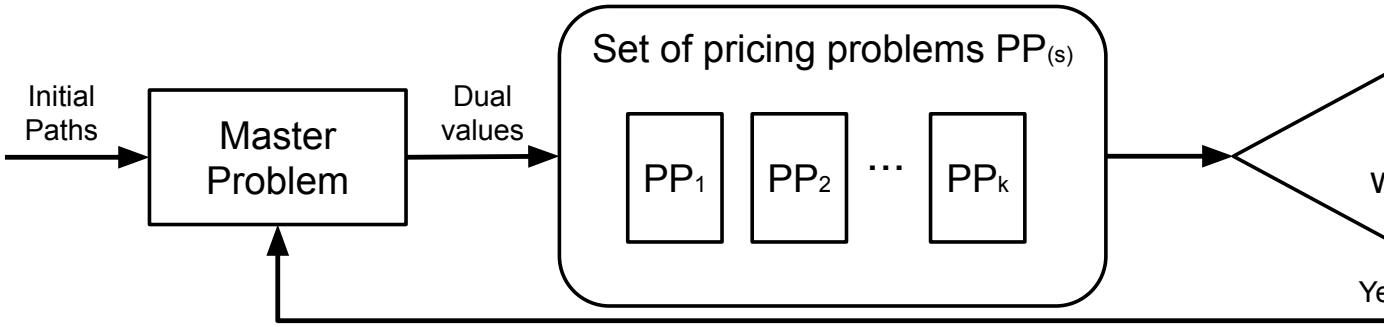


Figure 3: CG is a decomposition method dividing an optimization model into two parts: a master problem and a (set of) pricing problem(s) (PP). The restricted master problem (RMP) solves a fractional relaxation of the problem with a restricted set of columns. Then the PPs compute the best columns to be added, based on prices given by the dual variables of the RMP. The RMP and PP are then iteratively solved until no more columns can improve the solution of the RMP. Last, the original problem is solved with the integrality constraint using the columns of the RMP.

Reconfiguration - Routing modification constraints. To know if the routing of a demand d is modified on links between two consecutive time periods.
For $d \in D$, $\ell \in E$, $i \in \{0, ..., |c_d|\}$, $t \in \{1, ..., T\}$.

$$\omega^{d,t} \leq 1 + \varphi_{\ell,i}^{d,t} - \varphi_{\ell,i}^{d,t-1} \tag{12}$$

$$\omega^{d,t} \leq 1 + \varphi_{\ell,i}^{d,t-1} - \varphi_{\ell,i}^{d,t} \tag{13}$$

Reconfiguration - Allocation modification constraints. To know if the allocation of a demand d is modified on nodes between two consecutive time periods.
For $d \in D$, $u \in V$, $i \in \{0, ..., |c_d|\}$, $t \in \{1, ..., T\}$.

$$\omega^{d,t} \leq 1 + \alpha_{u,i}^{d,t} - \alpha_{u,i}^{d,t-1} \tag{14}$$

$$\omega^{d,t} \leq 1 + \alpha_{u,i}^{d,t-1} - \alpha_{u,i}^{d,t} \tag{15}$$

As we will see in Section 5, although effective, the compact ILP model `slow-rescue` does not scale on large networks or with many slices. We therefore propose an alternative using column generation: `rescue-ILP` and `rescue-LP` (for REconfiguration of network Slices with ColUmn gEneration with ILP or LP pricing).

4.2 Description of our CG-based algorithms: `rescue-ILP` and `rescue-LP`

4.2.1 Main ideas of column generation

Column generation (CG) is a model allowing to solve an optimization model without explicitly introducing all variables, see Figure 3 for an explanation. It thus often allows to solve larger instances of the problem than a compact model, in particular, with an exponential number of variables. In our model, the master problem (MP) seeks a possible global reconfiguration for all slices. It uses a path-formulation. Only a subset of potential paths is used for each slice in the restricted master problem (RMP). The set of paths is initialized to the paths used before reconfiguration. Each pricing problem (PP) then generates a new path for a request, together with the placement of the VNFs. During a reconfiguration, slices are migrated from one path to another. Note that, as the execution of each pricing problem is independent of the others, their solutions can be obtained in parallel.

4.2.2 Master Problem of `rescue-ILP` and `rescue-LP`

This master problem is used both by `rescue-ILP` and `rescue-LP`.
Variables:
• $\varphi_p^{d,t} \in [0,1]$ is the amount of flow of demand d on path p at time step t.
• $y_p^{d,t} \in [0,1]$ is the maximum amount of flow of demand d on path p between time step $t-1$ and t.
• δ_l^p is the number of times the link ℓ appears on path p.
• $\theta_{i,u}^p = 1$ if node u is used as a VNF on path p on layer i.

We assume an initial configuration is provided with fixed values for $\varphi_p^{d,0}$. The optimization model is written as follows.
Objective: minimize the amount of network resources consumed during the last reconfiguration time step T.

$$\min \sum_{d\in D}\sum_{p\in P_d}\sum_{\ell\in E} \mathrm{BW}_d\, \varphi_p^{d,T}\, \delta_\ell^p + \beta \sum_{u\in V^{\mathrm{VNF}}}\sum_{f\in F} c_{u,f}\, z_{u,f} \tag{16}$$

Constraints:
One path constraint. For $d \in D$, time step $t \in \{0,...,T\}$.

$$\sum_{p\in P_d} \varphi_p^{d,t} = 1 \tag{17}$$

Path usage over two consecutive time periods. For $d \in D$, $p \in P_d$, $t \in \{1,...,T\}$.

$$\varphi_p^{d,t} \le y_p^{d,t} \text{ and } \varphi_p^{d,t} \le y_p^{d,t-1} \tag{18}$$

Make Before Break - Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed C_u considering the resources used over two consecutive time periods. For $u \in V^{\mathrm{VNF}}$, $t \in \{1,...,T\}$.

$$\sum_{d\in D}\sum_{p\in P_d}\sum_{i=0}^{|c_d|-1} y_p^{d,t}\cdot \theta_{i,u}^p \cdot \mathrm{BW}_d \cdot \Delta_{f_i^{c_d}} \le C_u \tag{19}$$

Make Before Break - Link capacity constraints. The capacity of a link $\ell \in E$ is shared between each layer and cannot exceed C_ℓ considering the resources used over two consecutive time periods. For $\ell \in E$, $t \in \{1,...,T\}$,

$$\sum_{d\in D}\sum_{p\in P_d} \mathrm{BW}_d\, y_p^{d,t}\, \delta_\ell^p \le C_\ell. \tag{20}$$

Function activation. To know which functions are activated on which nodes in the final routing. For $u \in V$, $f \in F$, $d \in D$, $i \in \{0,...,|c_d|-1\}$,

$$y_p^{d,T}\, \theta_{i,u}^p \le z_{u,f_i^{c_d}}. \tag{21}$$

4.2.3 ILP Pricing Problem of `rescue-ILP`

The pricing problem searches for a possible placement for the slice. Since a reconfiguration can be done in several steps, a pricing problem is launched for each demand, at each time step.

Parameters:

• μ are the dual values of the master's constraints. The number written in upperscript is the reference of the master's constraints.
Variables:
• $\varphi_{\ell,i} \in [0,1]$ is the amount of flow on link ℓ in layer i.
• $\alpha_{u,i} \in [0,1]$ is the amount of flow on node u in layer i.

Objective: minimize the amount of network resources consumed for the demand d at time t.

$$\min \sum_{\ell \in E} \sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \mathrm{BW}(1 + \mu_{l,t}^{(20)}) + \mathrm{BW} \sum_{u \in V^{\mathrm{VNF}}} \mu_{u,t}^{(19)} \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \, \alpha_{u,i} - \mu_{d,t}^{(17)} + \beta \sum_{u \in V^{\mathrm{VNF}}} \sum_{f \in F} c_{u,f} \, z_{u,f} \, \mu_{d,u,f}^{(21)} \quad (22)$$

where $\mu_{d,u,f}^{(21)} = 0$ when $t \neq T$, see constraints (21). **Constraints:**
Flow conservation constraints for the demand d. For $u \in V^{\mathrm{VNF}}$.

$$\sum_{\ell \in \omega^+(u)} \varphi_{\ell,0} - \sum_{\ell \in \omega^-(u)} \varphi_{\ell,0} + \alpha_{u,0} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \quad (23)$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,|c_d|} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,|c_d|} - \alpha_{u,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \quad (24)$$

$$\sum_{\ell \in \omega^+(u)} \varphi_{\ell,i} - \sum_{\ell \in \omega^-(u)} \varphi_{\ell,i} + \alpha_{u,i-1} - \alpha_{u,i-1} = 0 \qquad 0 < i < |c_d| \quad (25)$$

Delay constraints. The sum of the link delays of the flow must not exceed the delay requirement of demand d.

$$\sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \Gamma_\ell \leq \gamma_d \quad (26)$$

Function activation. To know which functions are activated on which nodes. For $u \in V^{\mathrm{VNF}}$, $f \in F$, layer $i \in \{0, ..., |c_d| - 1\}$

$$\alpha_{u,i} \leq z_{u,f_i^{c_d}} \quad (27)$$

Location constraints. A node may be enabled to run only a subset of the virtual network functions. For $v \in V^{\mathrm{VNF}}$, $i \in \{0, ..., |c_d| - 1\}$, if the $(i+1)^{\mathrm{th}}$ function of c_d cannot be installed on u, we have

$$\alpha_{v,i} = 0. \quad (28)$$

4.2.4 LP Pricing Problem of `rescue-LP`

The difference between `rescue-ILP` and `rescue-LP` comes from the pricing problem, which is integer for `rescue-ILP` and fractional for `rescue-LP`. Indeed, the execution time of the CG algorithm is divided into the resolutions of: (1) the multiple PPs, (2) the multiple relaxations of the RMP, and (3) the ILP of the MP.

In our experiments, the time spent in (1) represents more than 90% of the whole execution time. To reduce this computational time, we propose `rescue-LP` that solves a relaxation of the pricing problem with fractional flows. The Master Problem of `rescue-LP` is the same as

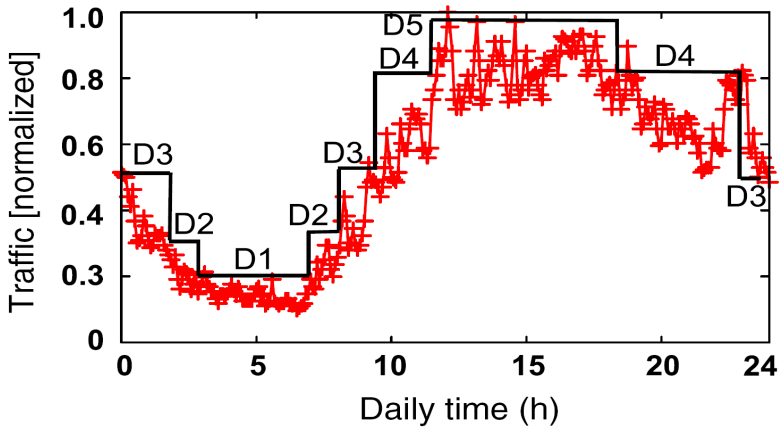


Figure 4: Period approximation of traffic variation

previously described. In the vast majority of cases, even with no constraint to force integral flows, the PP outputs an integral path that can be directly integrated into the RMP. If the LP gives a fractional flow, we use the ILP PP of `rescue-ILP` to get an integral path.

5 Numerical Results

We conducted several experiments in order to show the efficiency of our algorithms. In this section, we present first the data sets in Section 5.1. Then we present the results of our simulations showing the efficiency of our column generation models compared to `slow-rescue` in Section 5.2. We perform comparisons for different volume of traffic, and for several metrics. Then, we discuss the impact of the number of reconfiguration steps in Section 5.3. We then discuss the gain provided by the reconfiguration in Section 5.4 using a dynamic scenario, where requests arrive and leave over time. Then, we study in Section 5.5 the impact of reconfiguration frequency on several metrics: network operating costs, throughput, percentage of accepted slices, and cost per accepted Mbit of traffic. In Section 5.6, we push our algorithms to the limits with an high number of slices, on a big network, and show how parallelization can improve the running times. Finally, we show in Section 5.7 how the delay constraints imposed by the slices impact the network operating costs.

5.1 Data sets

We conduct simulations on three real-world topologies from SNDlib [19] of different sizes: `pdh` (11 nodes, 34 links), `ta1` (24 nodes, 55 links), and `ta2` (65 nodes, 108 links).

We consider four different types of slices corresponding to four services: Video Streaming, Web Service, VoIP, and online gaming. The characteristics of each service are reported in Table 2 and are taken from [20][21]. They differ in terms of VNF chains, bandwidth usage, and latency requirement. Each slice has to implement a chain of 5 VNFs and requires a specific amount of bandwidth. The latency requirements are expressed in terms of maximum stretch, i.e., the ratio between the path delay compared to the shortest path between the source and destination. Simulations have been conducted on an Intel Xeon E3-1271 v3 with 32GB of RAM.

Our goal was to study the impact of reconfiguration for different network usages. Indeed, when the traffic is low or medium, all slices can be served and reconfigurations improve the network

Slice Types	VNF chain	Latency	BW (Mbps)
Video Streaming	NAT-FW-TM-VOC-IDPS	High	256
Web Service	NAT-FW-TM-WOC-IDPS	Medium	100
VoIP	NAT-FW-TM-FW-NAT	Low	64
Online Gaming	NAT-FW-VOC-WOC-IDPS	Very low	50

Table 2: Characteristics of network slices

	pdh		ta1	
	rescue-ILP	rescue-LP	rescue-ILP	rescue-LP
D1	1.46	4.52	0.16	1.05
D2	10.27	9.69	3.74	5.41
D3	1.76	2.61	7.65	7.89
D4	12.26	10.87	10.67	10.86
D5	9.45	10.57	13.87	14.59

Table 3: Accuracy of the column generation models (%)

usage (links and VNFs). However, when the traffic is high and if some links are congested, reconfiguration also helps to prevent denying slices. To model the typical daily variation of traffic in an ISP network, we used the traffic distribution from a trace of the Orange network (Fig. 4). We adapted the churn rate of slices during time in order to obtain a similar level of traffic. This distribution is decoupled into five different traffic demands: D1 to D5, D1 being the minimum one (from 3 to 6 am) and D5 the highest one (from 11am to 6pm). Each level of traffic corresponds to a different average number of slices: from 30 for D1, 68 for D2, 105 for D3, 158 for D4 to 180 for D5 for `pdh`.

We evaluate and compare 5 different algorithms:

- `no-reconf` places and removes the slices without reconfiguring the network,
- `slice-wreck` reconfigures regularly the network, but with interruptions. When a reconfiguration is needed, it computes an optimal routing and placement solution, and reconfigures the slices to that new solution. This algorithm gives a bound of the best we can reach with the make-before-break approach,
- `slow-rescue`: our compact ILP that reconfigures slices without interruptions,
- `rescue-ILP`: our CG based algorithm with ILP pricing,
- `rescue-LP`: our CG based algorithm with LP pricing.

5.2 Efficiency of our algorithms with different traffic matrices

In this section, we consider the `pdh` and `ta1` networks for five different levels of traffic during the day, as shown in Fig. 4. One reconfiguration with two steps is achieved per traffic matrix. All the slices of the traffic matrix are placed and routed, and one reconfiguration is achieved to reroute the slices in order to improve the network usage.

5.2.1 Execution times

We report the execution times of a reconfiguration in two steps for `slow-rescue`, `rescue-ILP`, and `rescue-LP` in Figure 5. Each value is an average over 10 experiments. We set a time limit of one hour. For `pdh`, `slow-rescue` finds the optimal solution only for the period D1. For all the other ones, it reaches the time limit. However, it succeeded to find a feasible solution (which was already efficient) for all the time periods, as can be seen in Fig 6. For the larger network `ta1`, the compact ILP was not able to find any feasible solution, even for D1 with few slices. On the contrary, the execution times of the column generation models are a low (below 120 s for both networks for any time period). Moreover, the models scale well as their execution times increase in a linear way. `rescue-LP` needs from 2 s to around 45 s, while the execution times of `rescue-ILP` are between 10 s and 120 s. Thus, as expected, `rescue-LP` is a lot faster than `rescue-ILP`.

5.2.2 Gains in network cost

We now compare in Figure 6 the improvement in terms of network cost obtained after a reconfiguration for each time period for `pdh` and `ta1`. The network cost is the weighted sum between the number of VNFs and the link bandwidth. These two last parameters are plotted in Figure 7 and 8. As discussed above, `slow-rescue` is not considered for `ta1`, as it is not able to find any solution even with one hour of time limit. Recall also that `slow-rescue` provides an optimal value only for D1 for `pdh`. For the other traffic periods, the time limit is often reached. For `pdh`, we observe that the results of `rescue-ILP` and `rescue-LP` are very close to the ones of `slow-rescue`: the difference is about 5% for D1 and less than 1% for the other periods. It shows that the column generation models achieve very good results (see the accuracy of the models reported in Section 5.2.3), while being a lot faster than the ILP model.

For both networks, we see that `rescue-ILP` and `rescue-LP` achieve comparable results. As `rescue-LP` is faster, we use it as our *preferred solution* in the following. Last, we compare the results of our models with `slice-wreck`, which does not use the make-before-break mechanism. `slice-wreck` can achieve a better network improvement but at the cost of breaking slices and, thus, of a degraded QoS for users. We report its results as an upper bound on what our algorithms can achieve. We see that `rescue-ILP` and `rescue-LP` results are within few percent of the ones of `slice-wreck`, showing their efficiency. The difference is higher for heavy load periods (D4 and D5). Indeed, when the traffic is high, some links are almost saturated, it is harder to ensure that the bandwidth for both the current path and the path targeted by the reconfiguration can be reserved during the process.

5.2.3 Accuracy of the Column Generation Models

The accuracy ε of a column generation model is classically defined as $\varepsilon = (\tilde{z}_{\text{ILP}} - z^{\star}_{\text{LP}})/z^{\star}_{\text{LP}}$, where $z^{\star}_{\text{LP}}$ represents the optimal value of the relaxation of the Restricted Master Problem, and $\tilde{z}_{\text{ILP}}$ the integer solution obtained at the end of the column generation algorithm. We provide the accuracy of `rescue-ILP` and `rescue-LP` in Table 3. We see that, if the accuracy increases with the number of slices, it is always better than 15% for both networks. The solutions thus are not far from optimal.

5.2.4 Time limits for the reconfiguration

The reconfiguration of the network has to be done dynamically in real time. In this context, the time to compute the reconfiguration is an important element towards the adoption of such

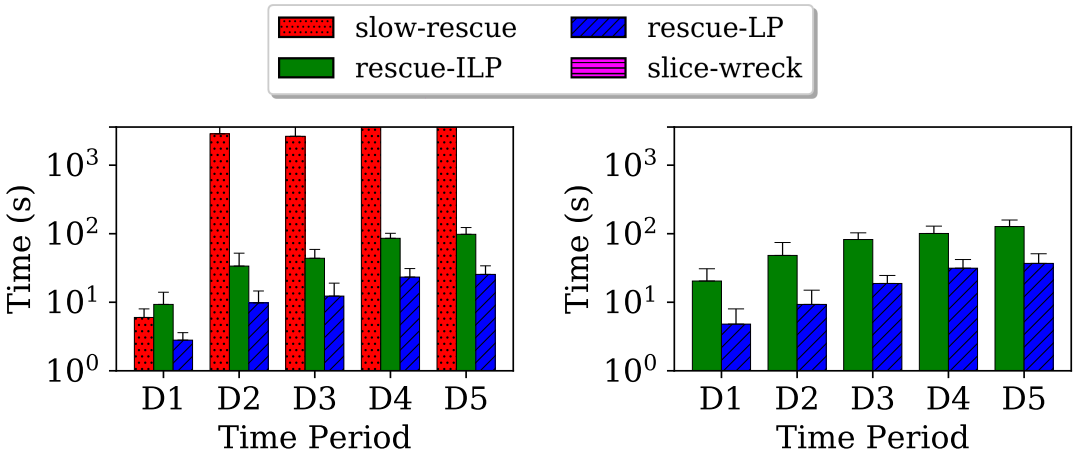


Figure 5: Execution times for `pdh` (left) and for `ta1` (right).

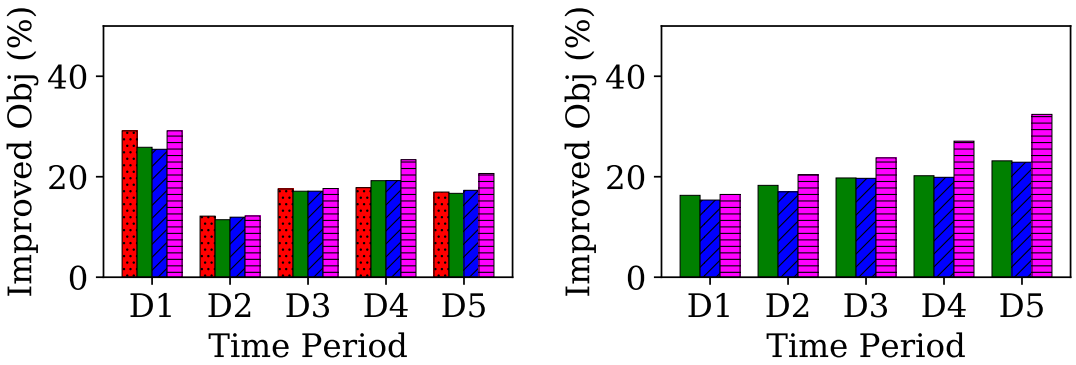


Figure 6: Gains in network cost for `pdh` (left) and for `ta1` (right).

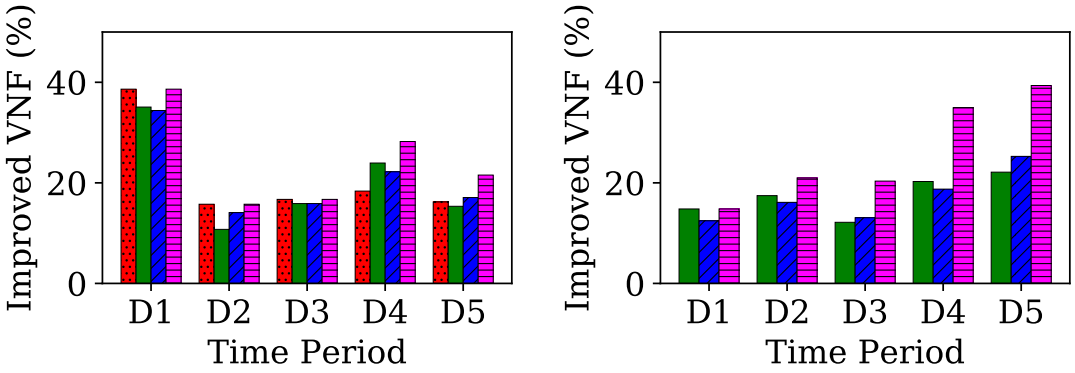


Figure 7: Gains in VNF for `pdh` (left) and for `ta1` (right).

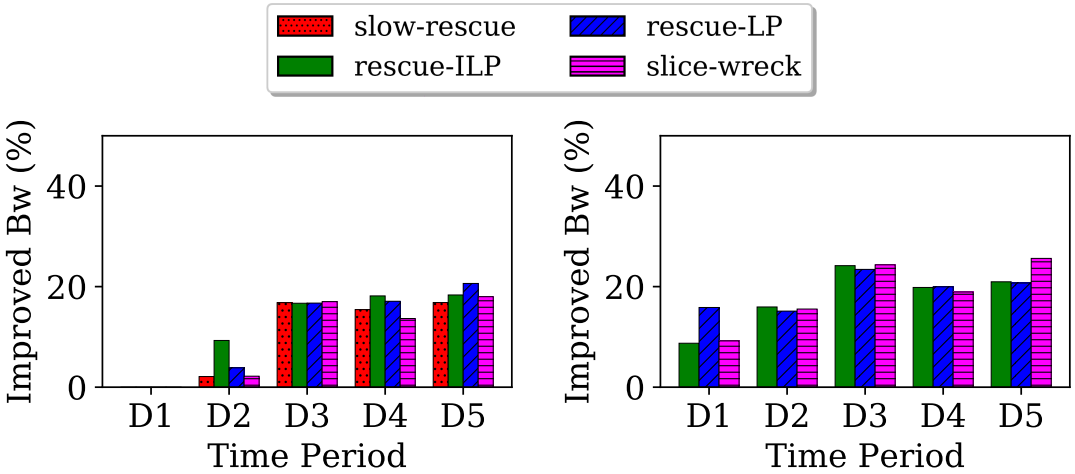


Figure 8: Gains in bandwidth for `pdh` (left) and for `ta1` (right).

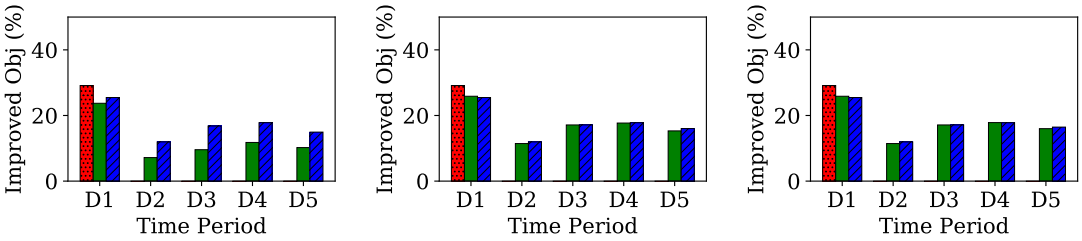


Figure 9: Improvement due to the reconfiguration for different model time limits: 10 s (left), 60 s (middle) and 600 s (right) on `pdh`.

solutions. We thus compare the results of the algorithms for `pdh` for different maximum execution times: 10, 60 and 600 seconds, see Figure 9. The first observation is that `slow-rescue` only gives a solution for D1, even with 600 s of execution. Secondly, `rescue-ILP` takes at least 60 s to reach its best value for any time period while `rescue-LP` reaches it in 10 s for all periods (except for D5 where it needs 60 seconds to reach it). It confirms that `rescue-LP` is the most scalable method while reaching similar performance as `rescue-ILP`. It thus is the best solution to use in practice.

5.3 Impact of the number of reconfiguration steps

The main specificity of our *make-before-break* scheme is that the reconfiguration is done in a given number of steps. In this section, we are interested in the impact of the number of steps of the reconfiguration on the same scenario of previous section: one reconfiguration for each traffic matrix D1 to D5. The simulations are done on `ta1` with a varying number of steps from 1 to 3.

5.3.1 Network cost

In Figure 10, we observe that two steps of reconfigurations lead to better network cost improvement than with only one step. Indeed the one-step reconfiguration is very constraining: some slices need to be moved to an intermediate position before reaching their final one. Therefore one-step only allows an improvement of around 10% on average while it is around 18% for two steps. The difference between 2 and 3 steps is significant for periods D4 and D5 (heavy traffic where one additional step is needed to move the slices) where we have from 2.5% to 6% of additional improvement. During these two last periods, `slice-wreck` behaves well, as it is able to interrupt slices and therefore has no constraints for reconfiguration.

5.3.2 Effect on execution time

The increase in the number of steps is not only beneficial, as can be seen in Figure 11, it is also linked to an increase in the computation time. As we saw in section 4.2.3, we run a pricing for each request and for each step. A reconfiguration in 3 steps will therefore have 3 times more pricing to solve than with 1 step. Also, there are additional paths to be processed in the restricted Master problem. In period D5, for `rescue-LP` we go from 16 s in 1 step to 32 s and 72 s to 2 and 3 steps. For `rescue-ILP` we are still above it with 77 s in 1 step, 107 s in 2 steps and 232 s in 3 steps.

5.3.3 Effect on the percentage of reconfigured slices

The second detrimental effect of increasing the number of steps is the number of modified slices. As we can see in Figure 12, the number of reconfigured slices increases with the number of reconfiguration steps. Note that `slice-wreck` does not include reconfiguration steps, and therefore the purple box-plots for `slice-wreck` are the same for the three figures. The improvement of the network cost for `slice-wreck` is done at the cost of reconfiguring and interrupting more than half of the slices at each phase of reconfiguration. Therefore, at least 50% of the slices will experience a degradation of the *QoS* since `slice-wreck` interrupts the slices.

5.4 Gains over Time

In the following, we are considering a scenario where reconfiguration is regularly performed, and where traffic is dynamic (requests arrive and leave during time). We now study the gains provided by the reconfiguration during time. To this end, we compare the results of `rescue-LP`

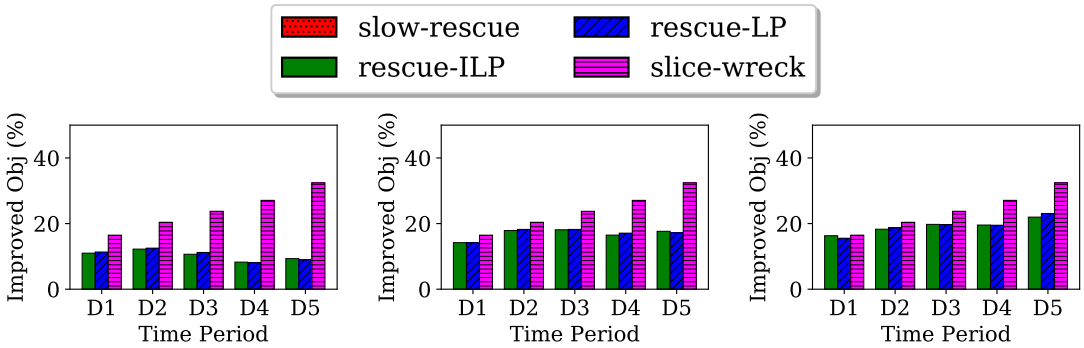


Figure 10: Improvement of the Objective (in %) with 1 step (left), 2 steps (middle) and 3 steps (right) of reconfiguration on `ta1`.

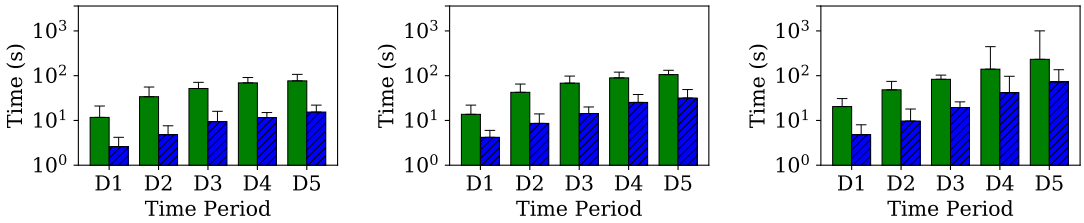


Figure 11: Reconfiguration time with 1 step (left), 2 steps (middle) and 3 steps (right) of reconfiguration on `ta1`.

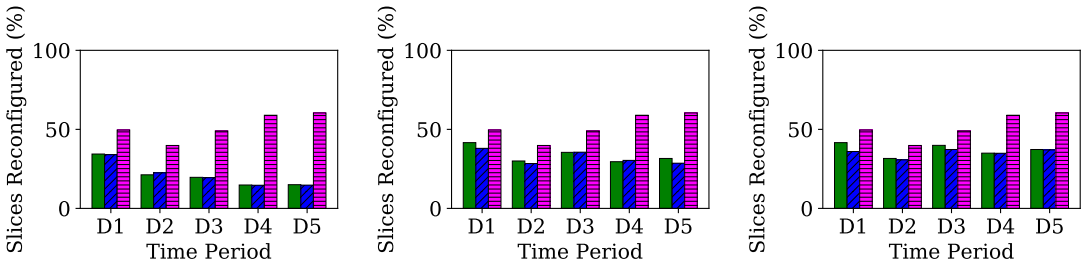


Figure 12: Percentage of slices reconfigured with 1 step (left), 2 steps (middle) and 3 steps (right) of reconfiguration on `ta1`.

with the ones of `no-reconf` which does not reconfigure the slices. We consider in this section two networks: the medium and large, `ta1` and `ta2`. In our scenario, the network has periods of high congestion during which some slices may be rejected. We thus study the following metrics: the network operational cost, the throughput of the accepted slices, the accepted number of slices, and the operational cost per MBits of accepted traffic. . `rescue-LP` performs reconfigurations every 15 minutes. We choose this value as it seems a reasonable one for a network operator which does not want to change its routes too frequently. This choice is also discussed in next section (Section5.5) where we vary the reconfiguration frequency, and show that 15 is a good trade-off between network management and all the studied metrics.

5.4.1 Network Cost

In Figure 13 we study the network operational cost over time. The network cost follows the traffic variation depicted in Figure 4: the more traffic, the more network operational cost. Our solution is more reactive to traffic variations thanks to the reconfigurations that are regularly performed. Throughout the entire execution and for both networks, `rescue-LP` reduces significantly the network operational costs: 21% of reduction on `ta1` and 18% on `ta2` compared to `no-reconf` case. This reduction is particularly interesting when the network is loaded (between 10am and 6pm). This implies that the network is better managed and the resources are used more efficiently.

5.4.2 Throughput

The objective of our solution is to reduce operational costs. However, we should not reduce these costs at the price of rejecting slices. Therefore, we present the global throughput of the network in Figure 14. This throughput is defined as the sum of the requested bandwidth of the accepted slices. As we can see on both networks, `ta1` and `ta2`, during the first 5 hours of execution there is almost no congestion because the traffic decreases. Therefore, `no-reconf` and `rescue-LP` accept the same number of slices, and therefore get roughly the same throughput. The next 3 hours, traffic increases and `rescue-LP` improves the throughput until almost 10% on `ta2` when the network is the most saturated (traffic period D5). For a period of 24 hours, `rescue-LP` allows an average throughput improvement of 4% on `ta1` and 7% on `ta2`. Therefore, as a conclusion on these two last figures, `rescue-LP` reduces network operational costs while at the same time improving the network throughput. These gains are reached without impacting users' Quality of service.

5.4.3 Accepted Slices

The difference in terms of throughput discussed above comes from different slice acceptance rates of both solutions. As the slices of different types do not require the same reserved bandwidth, we report the percentage of the bandwidth of the accepted slices compared to the one of the requested slices. The curves in Figure 15 represent incremental acceptance. It obviously starts at 100% acceptance. Then, we can see the same general tendency for both networks, but more pronounced for `ta2`: a decrease of the acceptance at the beginning during the period D3, then an increase when the traffic is lower (D2 and D1), and finally a decrease for the period D4 and D5 when heavy congestion. The first decrease at the beginning of the curve on `ta2` is due to a small congestion in period D3 (that occurs at 7am) which does not allow to accept 100% of the slices. `rescue-LP` allows an improvement in slice acceptance for both networks: 2.5% and 5% more bandwidth for `ta1` and `ta2`, respectively.

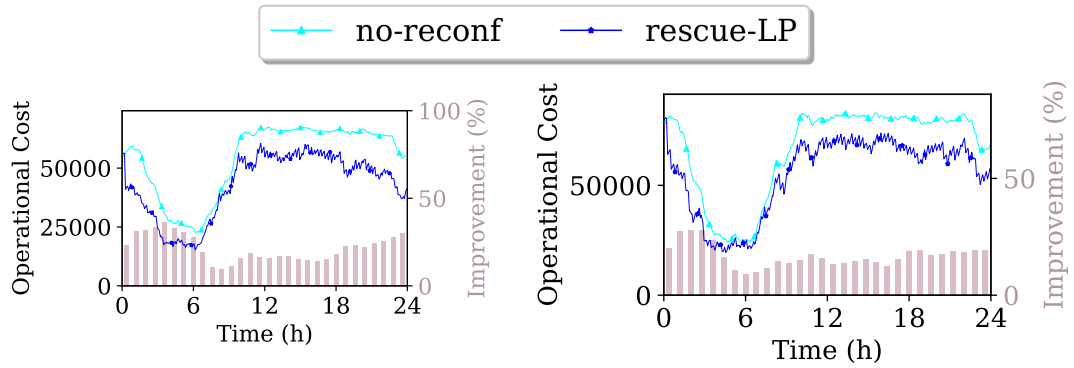


Figure 13: Network cost for `ta1` (left) and for `ta2` (right).

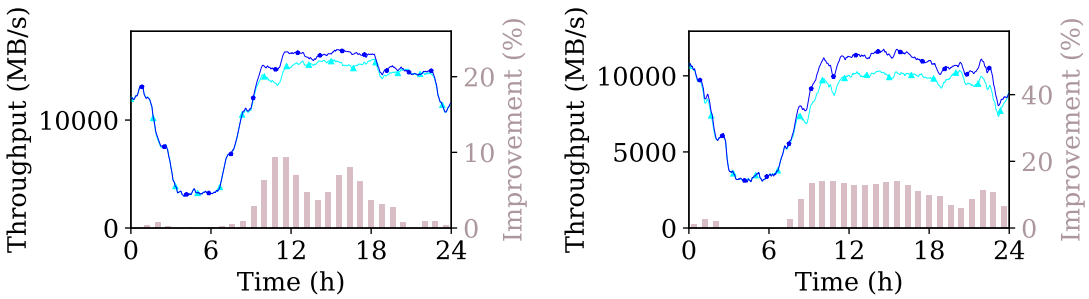


Figure 14: Throughput for `ta1` (left) and for `ta2` (right).

5.4.4 Cost per MBit

As discussed above, reconfiguration allows to reduce the network operational cost and, at the same time, to accept more slices. To measure both advantage with a single metric, we report the cost per MBit to obtain a fair comparison in Figure 16. The improvement in percent is given by the light red bars. The gain is of 25% for `ta1` and 23% for `ta2`. This shows that our solution is significantly efficient. We observe that the gain is lower when the traffic is low (period D1), but similar for the other periods (D2, D3, D4, D5). We also see that reconfiguring the network keeps the cost per MBit more stable during time, showing a better usage of the network resources which adapt when the traffic varies.

5.5 Impact of the reconfiguration time interval

In the previous section, we measured the effects of regularly reconfiguring the network in a dynamic scenario. The reconfiguration interval was set to 15 minutes. We now study the effects of different reconfiguration frequencies: 5 , 15 , 30 , and 60 min. Indeed, reconfiguring more regularly can improve the usage of the network resources, but at the same time lead to more difficult management. Reconfiguring less regularly eases management, but reduces the reconfiguration gains.

5.5.1 Network Cost

We study in Figure 17 the network operational cost of the network considering different reconfiguration frequencies. For frequency of 60 , 30 , 15 and 5 respectively, we have improvements

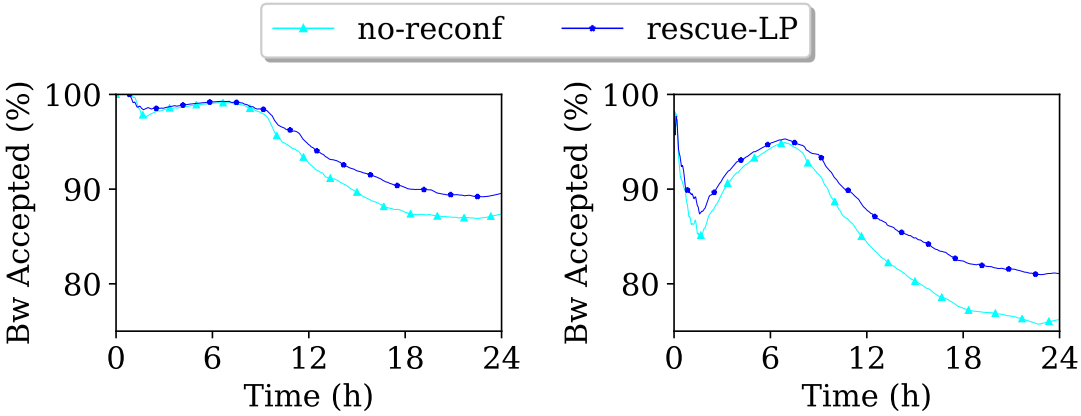


Figure 15: Percentage of Bandwidth accepted for `ta1` (left) and for `ta2` (right).

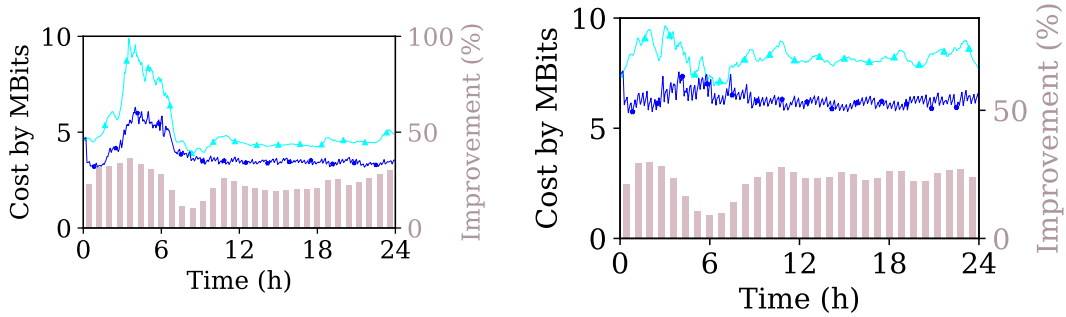


Figure 16: Network cost per accepted bandwidth for `ta1` (left) and for `ta2` (right).

of 15.8%, 18.7%, 20.7% and 22.7% on `ta1` and 11.4%, 14.9%, 17% and 21.7% on `ta2`. Even if a frequency of 5 leads to better improvement in network costs, good improvement is already obtained with a reconfiguration frequency of 60 , meaning a reconfiguration every hour.

5.5.2 Throughput

Figure 18 shows the network throughput over time as defined in 5.4.2. For reconfiguration frequency of 60 , 30 , 15 and 5 respectively, there are improvements of 1.1%, 1.9%, 3.5% and 4.2% on `ta1` and 3.9%, 6.5%, 8.9% and 9.9% on `ta2`. For both networks, a reconfiguration frequency every 15 minutes seems to be a good trade-off between throughput and network management.

5.5.3 Accepted Slices

In Figure 19, we plot the accepted bandwidth over time as defined in 5.4.3. Each curve is more easily identifiable compared to previous figures. For reconfiguration frequency of 60 , 30 , 15 and 5 respectively we have improvements of 0.6%, 1.3%, 2.4% and 3% on `ta1` and 2.4%, 4.6%, 6.4% and 7.6% on `ta2`. Here again, reconfiguring every 15 minutes seems to be a good trade-off for the accepted number of slices.

5.5.4 Cost per MBit

Figure 20 shows the network operational cost per MBit over time as defined in 5.4.4. We can easily distinguish the above curve without reconfiguration among all the curves. For reconfiguration frequency of 60 , 30 , 15 and 5 respectively there are improvements of 17.6%, 21.6%, 25.5% and 27.5% on `ta1` and 14.6%, 19.5%, 23.2% and 28% on `ta2`. Reconfiguring once an hour leads to strong peaks of cost, while when we reconfigure every 5 minutes, the cost per Mbit is more stable.

To summarize, a reconfiguration frequency of 15 is a good trade-off to balance the cost, stability and ease of network management. It leads to an improvement of 20.7% (respectively 17%) of network cost, 3.5% (respectively 8.9%) of throughput, 2.4% (respectively 6.4%) of accepted bandwidth, and of 25.5% (respectively 23.2%) of cost per Mbit on `ta1` (respectively on `ta2`).

5.6 Scalability and Parallelization

5.6.1 With a large number of slices

In this part we look at Figure 21 and Figure 22 and we are interested in the scalability of our solution based on our experiences in 5.2. Here we modify the load of the periods D1 to D5 for which there are now between 50 and 300 slices on the left side of the figures and between 85 and 500 slices on the left side of the figures. We impose a maximum time of 60 seconds. Note that only `rescue-ILP` and `rescue-LP` are compared, and recall that `slow-rescue` did not find any feasible solution with 2 steps of reconfiguration, with less than 30 slices in 3600 seconds on `ta1` (Figure 9(right)). For each of the networks `ta1` and `ta2` we perform a 3-steps reconfigurations using `rescue-ILP` and `rescue-LP`. As we can see, even with a large number of slices and a limited time, our solution still allows a significant improvement of the objective. Figure 21 shows us the results on `ta1` where `rescue-ILP` improves on average by 14.4% for 300 slices and 9.8% for 500, while `rescue-LP` goes from 17.5% to 12.9%. Figure 22 shows the results on `ta2` where `rescue-ILP` improves on average by 22.1% for 300 slices and 15.9% for 500, while `rescue-LP` goes from 26.5% to 22.8%. We can notice that the improvement is greater on `ta2`, as well as the decrease in improvement is less for `ta2` which is bigger and with a larger diameter than `ta1`. Finally we can see here the advantage of `rescue-LP` over `rescue-ILP` which allows a better

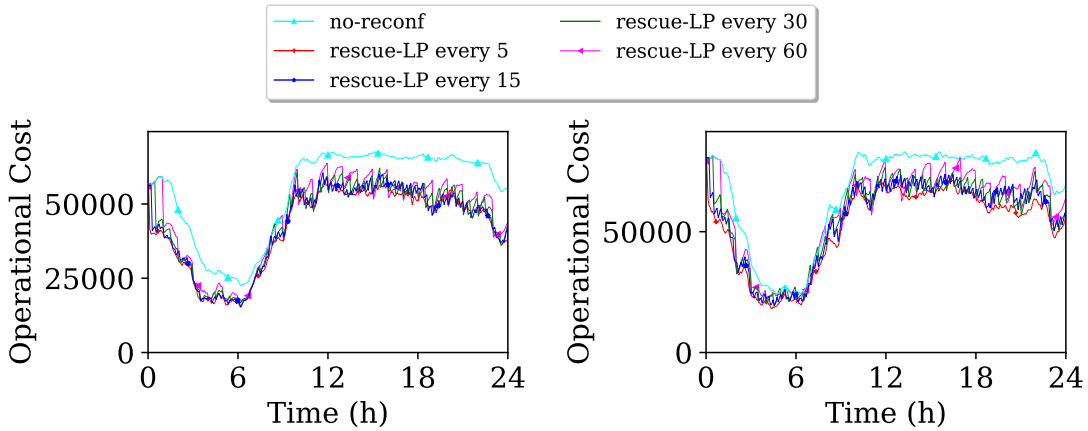


Figure 17: Network cost for `ta1` (left) and for `ta2` (right).

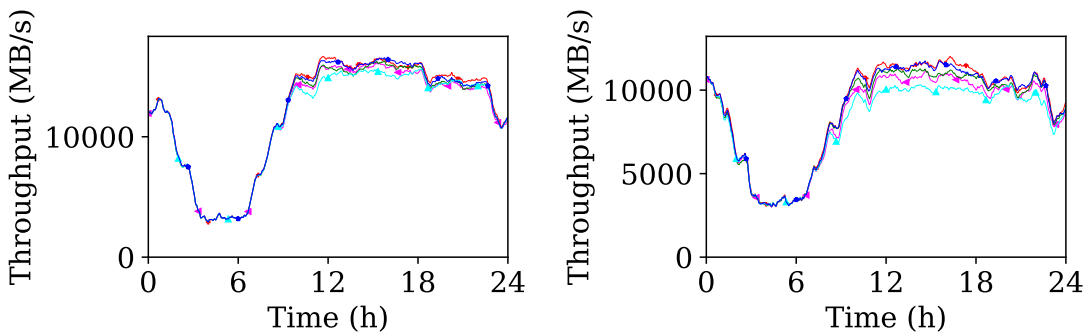


Figure 18: Throughput for `ta1` (left) and for `ta2` (right).

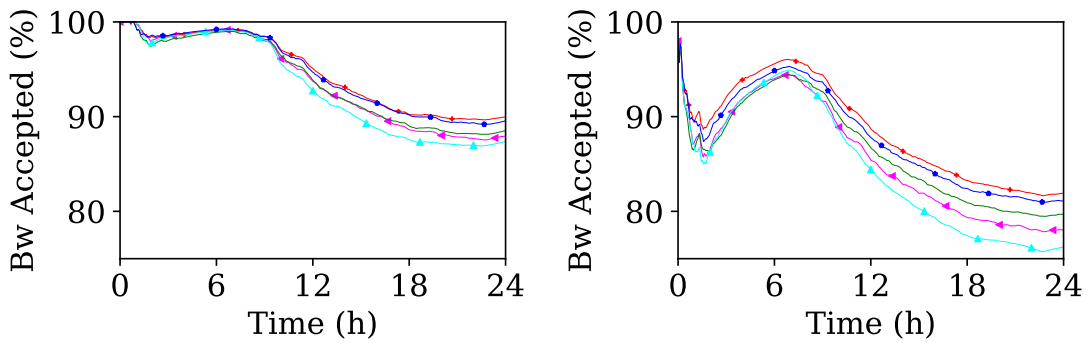


Figure 19: Percentage of Bandwidth accepted for `ta1` (left) and for `ta2` (right).

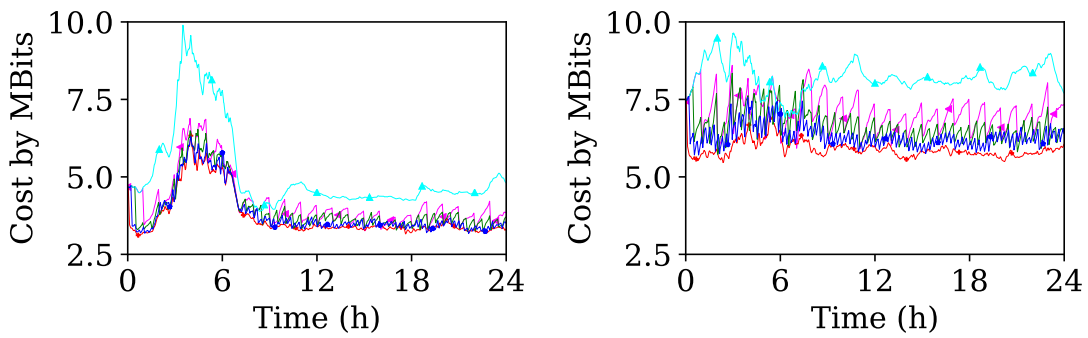


Figure 20: Network cost per accepted bandwidth for `ta1` (left) and for `ta2` (right).

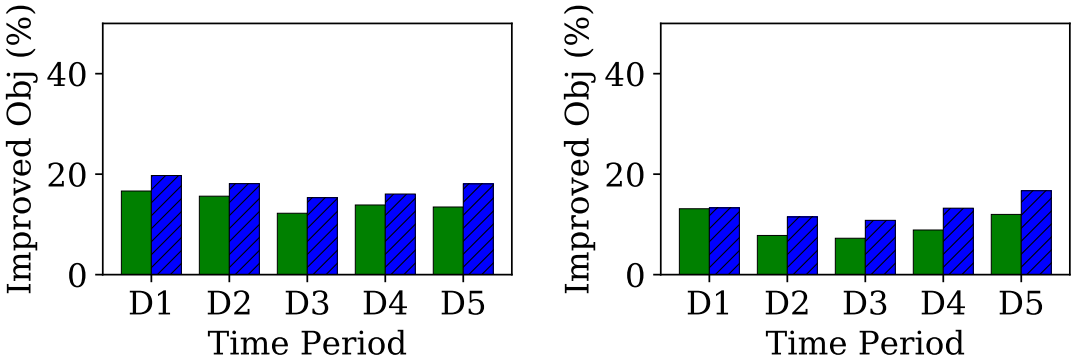


Figure 21: Gains in network cost for `ta1` with 300 slices (left) and with 500 slices (right).

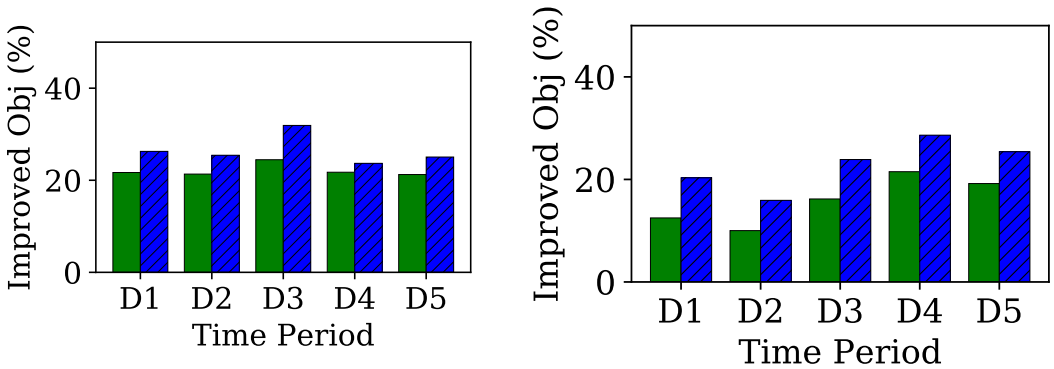


Figure 22: Gains in network cost for `ta2` with 300 slices (left) and with 500 slices (right).

improvement and is less affected by the lack of time on large instances, as we can see on `ta2` where the gap between the two widens from 4.4% with 300 slices to 6.9% with 500.

5.6.2 Parallelization of the pricing problem

One of the advantages of column generation is the ability to parallelize the execution of pricing problems on several CPUs cores or machines. In our experiments about 70% of the execution time is spent on solving pricing problems, which means that parallelization can save time. In Figure 23 we show the execution times of `rescue-LP` to reconfigure 400 slices in `ta2`. The average computation time is 530 seconds with 1 thread and 290 seconds with 2 threads (45% improvement). With 4 threads `rescue-LP` is faster and computes a solution on 185 seconds. The difference between 4 and 8 threads is negligible, 9 seconds less, but our computer, although having 8 threads, has only 4 CPU cores. As pricing execution already uses CPUs to their full potential, additional threads have only an extremely limited impact.

5.7 Impact of the delay constraints

Being able to ensure strict delay constraints for some applications is one of the key element of network slicing [4]. We thus study the impact of different delay constraints on the reconfiguration gains. The latency required by each slice type was categorized into four main categories: Very

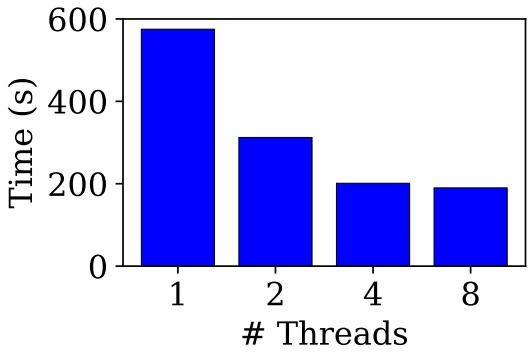


Figure 23: Time to execute the pricing problems according to the number of threads on `ta2` with 400 slices.

Low, Low, Medium or High, see Table 2. We carried out reconfigurations for `ta1` setting the delay constraints of each slice successively to each level.

5.7.1 Delay make it harder

In Figure 24 we can see the difference in the improvement of the objective according to the latency constraints. Reconfiguring slices with very low latency while keeping this constraint is hard, no more than 7% of improvement of network cost on average. By increasing latency, however, we have more possible paths and get a better improvement. The improvement reaches 27% with normal constraints and 40% if all slices have high latency. The most significant improvements are in case of low traffic (D1 and D2), for which the network has more residual capacity.

5.7.2 Delay make it faster

In Figure 25 we study the time of these reconfigurations. The more constraints on latencies the fastest to compute a reconfiguration. Indeed, with low latencies constraints, few paths are possible for the slices, while with high latencies there is a large set of possible paths, increasing the computation time..

6 Conclusion

In this work, we provide solutions, `rescue-ILP` and `rescue-LP`, to reconfigure a set of requests using a make-before-break approach. Our algorithms, based on column generation, reroute the requests to an optimal or close to optimal solution without impacting the rerouted requests. `rescue-LP` is the solution to be chosen in practice as we observed during simulations that it scales better with the network and the number of slices. Reconfiguring regularly the network with `rescue-LP` allows a slight increase in throughput when the network is congested as well as a significant reduction in operating costs of around 20%.

References

[1] M. Chiosi *et al.*, "Network functions virtualisation (NFV) network operator perspectives on industry progress," in *SDN and OpenFlow World Congress*, 2013.

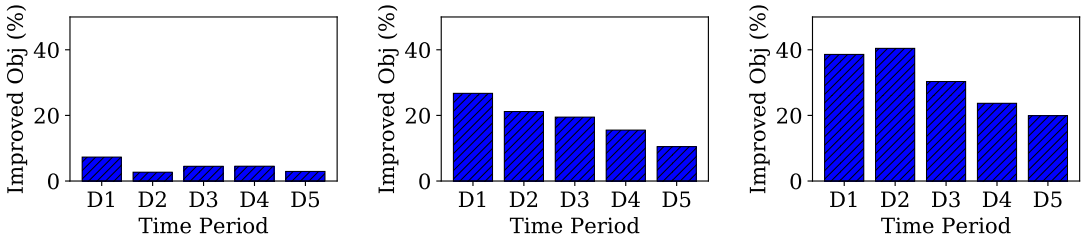


Figure 24: Improved Objective with low latencies (left), normal latencies (middle) and high latencies (right) reconfiguration on `ta1`.

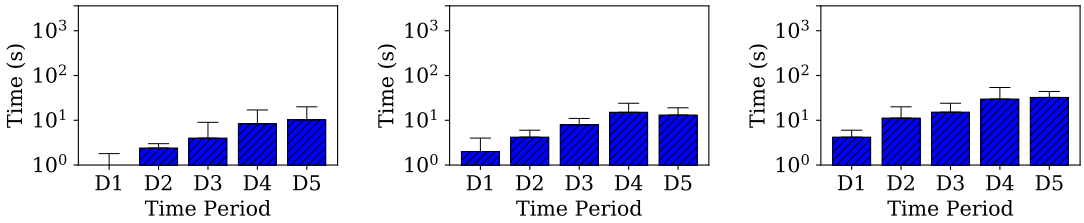


Figure 25: Reconfiguration time with low latencies (left), normal latencies (middle) and high latencies (right) reconfiguration on `ta1`.

[2] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, 2013.

[3] P. Rost *et al.*, "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Communications magazine*, 2017.

[4] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *IEEE INFOCOM*, 2017.

[5] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE TNSM*, 2016.

[6] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, 2016.

[7] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 4, pp. 1562–1576, 2018.

[8] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015.

[9] N. Huin, B. Jaumard, and F. Giroire, "Optimal network service chain provisioning," *IEEE/ACM Transactions on Networking*, 2018.

[10] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE INFOCOM*, 2018.

[11] R. Wang and B. Mukherjee, "Provisioning in elastic optical networks with non-disruptive defragmentation," *IEEE Journal of Lightwave Technology*, 2013.

[12] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE TNSM*, 2017.

[13] K. A. Noghani, A. J. Kassler, and J. Taheri, "On the cost-optimality trade-off for service function chain reconfiguration," in *IEEE International Conference on Cloud Networking (CloudNet)*, 2019.

[14] L. Gao and G. N. Rouskas, "Virtual network reconfiguration with load balancing and migration cost considerations," in *IEEE INFOCOM*, 2018.

[15] G. Wang, G. Feng, T. Q. Quek, S. Qin, R. Wen, and W. Tan, "Reconfiguration in network slicing-optimizing the profit and performance," *IEEE TNSM*, 2019.

[16] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac, "Don't Interrupt Me When You Reconfigure my SFCs," in *IEEE International Conference on Cloud Networking (CloudNet)*, 2019.

[17] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. C. Kozat, "A resource allocation framework for network slicing," in *IEEE INFOCOM*, 2018.

[18] M. Pozza, A. Patel, A. Rao, H. Flinck, and S. Tarkoma, "Composing 5G network slices by co-locating VNFs in μslices," in *IFIP Networking Conference*, 2019.

[19] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0–survivable network design library," *Wiley Networks*, 2010.

[20] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing costs on service chain placement in network functions virtualization," in *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015.

[21] *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019*, CISCO, May 2015.

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803